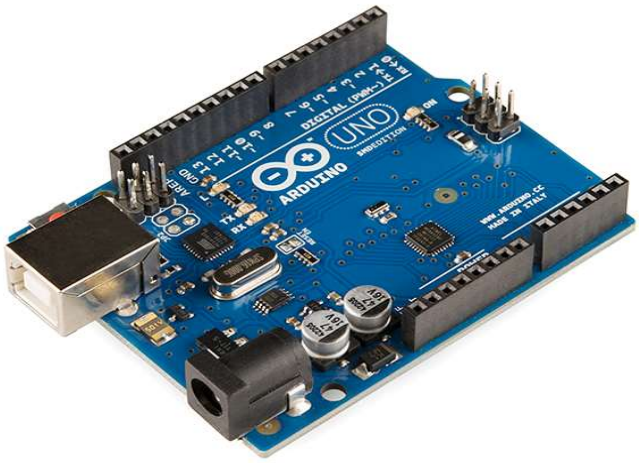
ARDUINO:

INTRODUCCIÓN A LA PROGRAMACIÓN

ARDUINO™

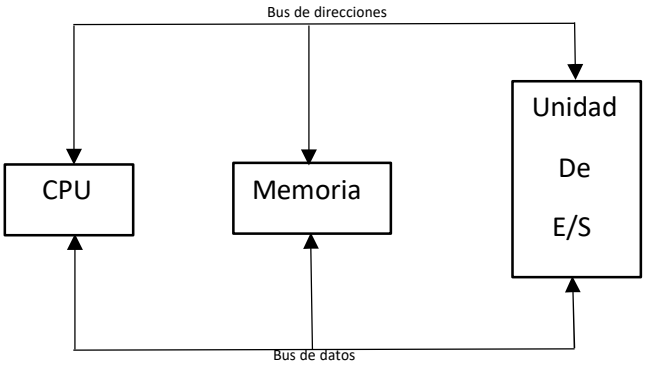


ÍNDICE

MICROCONTROLADOR

Un microcontrolador es un circuito integrado programable capaz de ejecutar una serie de instrucciones pregrabadas en su memoria. Podemos decir pues, que un microcontrolador es un computador en miniatura al cual le podemos ordenar la ejecución de rutinas repetitivas. Podemos diferenciar un microcontrolador frente a otros computadores embebidos con la particularidad de que este dispone de periféricos dedicados, es decir podemos realizar tareas tales como la lectura de valores analógicos de tensión, la activación de entradas y salidas, generar señales PWM, realizar comunicaciones, etc.

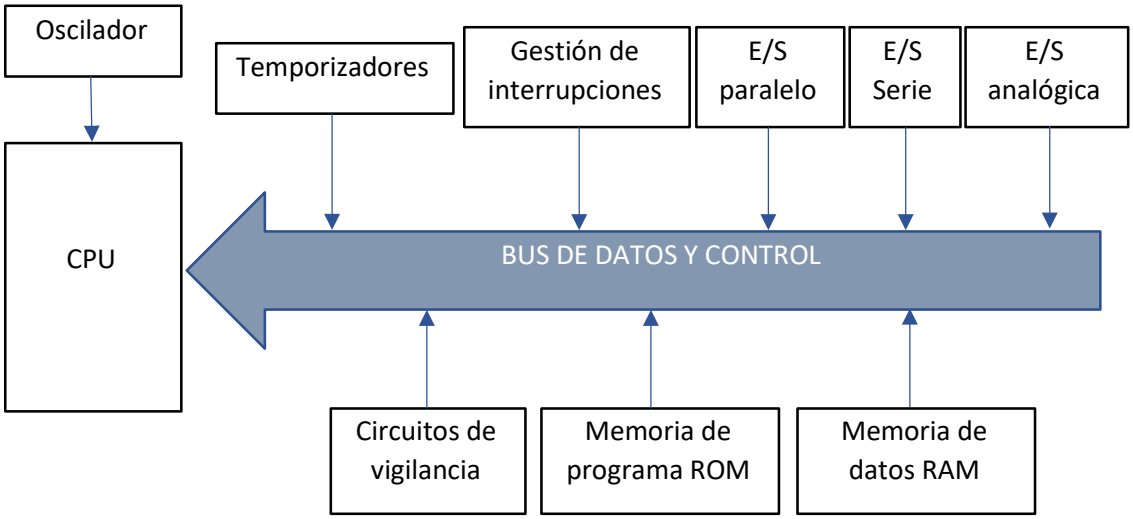
Un microcontrolador al uso incluye en su interior tres unidades funcionales principales: unidad central de procesado (CPU), memoria y unidades de entradas y salidas.



La CPU se encarga de la lectura y ejecución de las instrucciones almacenadas en la memoria de programa. La unidad de memoria se emplea para el almacenamiento de datos volátiles, persistentes y las instrucciones del programa. Por último, la unidad de entradas y salidas es la encargada de interactuar con el entorno.

Los microcontroles son útiles para obtener una solución compacta y de bajo coste que permita el control de procesos repetitivos o que evolucionan por medio de eventos medibles o predecibles. Así mismo los microcontroladores incorporar una serie de mecanismos que les permiten detectar anomalías en el funcionamiento del programa (watchdog), además de modos de funcionamiento de bajo consumo, protección de la propiedad intelectual de los programas y mecanismos hardware de atención a eventos asíncronos.

Componentes elementales de un microcontrolador:



Oscilador: proporciona un tren de pulsos constante con una frecuencia determinada que sirve como base de tiempo para la ejecución de instrucciones y los temporizadores internos.

Temporizadores y contadores: permiten el conteo de pulsos de una señal digital determinada o la medida del tiempo en unidades del sistema internacional.

Gestión de interrupciones: permiten capturar eventos externos asíncronos por hardware.

E/S paralelo: puerto con entradas y/o salidas digitales para la lectura de señales o la ejecución de una acción determinada.

E/S serie: Permite la comunicación o intercambio de datos con otros dispositivos presentes en el entorno. En un mismo microcontrolador podemos disponer de unidades que permitan tanto la comunicación serie síncrona como asíncrona.

E/S analógica: Permite la lectura o generación de señales analógicas.

Circuitos de vigilancia: se encargan de monitorizar en tiempo real la ejecución del programa y el estado del microcontrolador y avisarnos en caso de que se produzca alguna anomalia.

Memoria de programa (ROM): memoria no volátil en la que se almacenan las instrucciones preprogramadas para que sean ejecutadas por el microcontrolador.

Memoria de datos (RAM): Memoria volátil en la cual se almacenan de forma dinámica las variables y datos necesarias para la ejecución del programa. Esta memoria se reinicia con cada arrancada del microcontrolador.

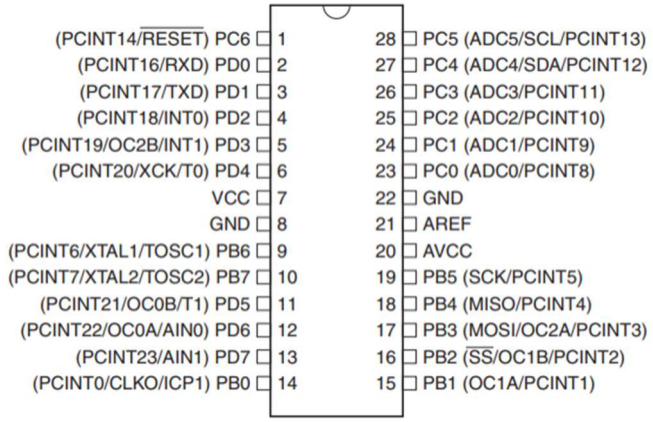
La familia de tarjetas de desarrollo Arduino incorpora diferentes microcontroladores para cada modelo que van desde microcontroladores AVR de 8 bits hasta procesadores ARM INTEL CORTEX M3. Las arquitecturas AVR y ARM no son a priori compatibles, así como los juegos de instrucciones de los diferentes microntroladores que se incluyen dentro de la familia Arduino, sin embargo, el entorno de desarrollo de Arduino está pensado para que se puedan usar indistintamente sus diferentes versiones de placas, haciendo así mismo que sean compatibles entre ellas. Esto se consigue mediante la definición de una serie de funciones elementales que permiten operar con prácticamente todos los periféricos presentes en un microcontrolador y que podemos conocer como juego de instrucciones de Arduino.

El entorno de desarrollo sobre el cual podemos editar, compilar y cargar los programas en las tarjetas de desarrollo de la familia Arduino se conoce como Arduino IDE. El entorno de desarrollo para Arduino es de libre acceso y se encuentra disponible tanto para sistemas MS-DOS como UNIX a través del siguiente enlace: https://www.arduino.cc/en/software. El IDE de Arduino es un entorno basado en processing que permite implementar programas diseñados en C++ bajo el lenguaje de alto nivel Wiring.

ATMEGA 328P

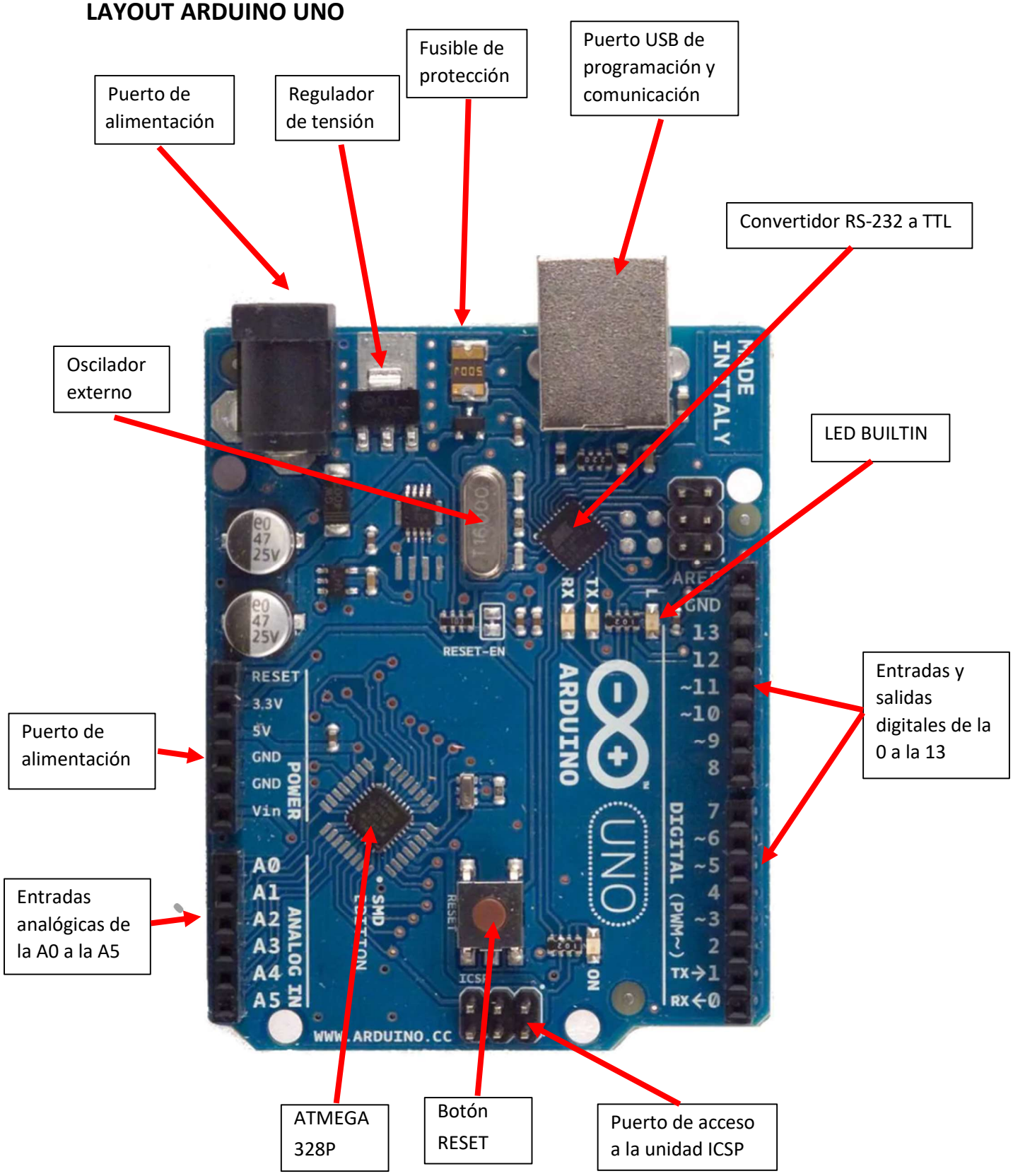
La tarjeta de desarrollo que emplearemos en el desarrollo de este curso es la conocida Arduino UNO. La tarjeta Arduino UNO incorpora el microcontrolador AVR de 8 bits ATMEGA 328P.

El microcontrolador ATMEGA 328P se trata de un microcontrolador de alto rendimiento AVR con juego de instrucciones RISC. Dispone de una memoria de programa Flash de 32KB programable por medio de una unidad ICSP. Dispone de una memoria no volátil tipo EEPROM para el almacenaje de datos de 1024B con 23 líneas de entrada y salidas y 32 registros de trabajo. Cuenta con dos temporizadores internos que podemos emplear para el contaje de pulsos, la medida de tiempo o la generación de señales tipo PWM por medio de su unidad de captura y comparación. Dispone así mismo de interrupciones externas programables, una unidad de comunicación serie asíncrona USART, dos unidades de comunicación serie síncrona SPI y una unidad serie síncrona I2C. Cuenta con un convertidor analógico a digital con 6 canales de 10 bits y tres puertos de entrada y salida digital. Puede operar con voltajes que van desde los 1.8V hasta los 5.5V a una frecuencia de oscilador máxima de 20MHz. Para disponer de una información más detallada puede consultarse las hojas de datos técnicas accesibles a través del siguiente enlace: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf.

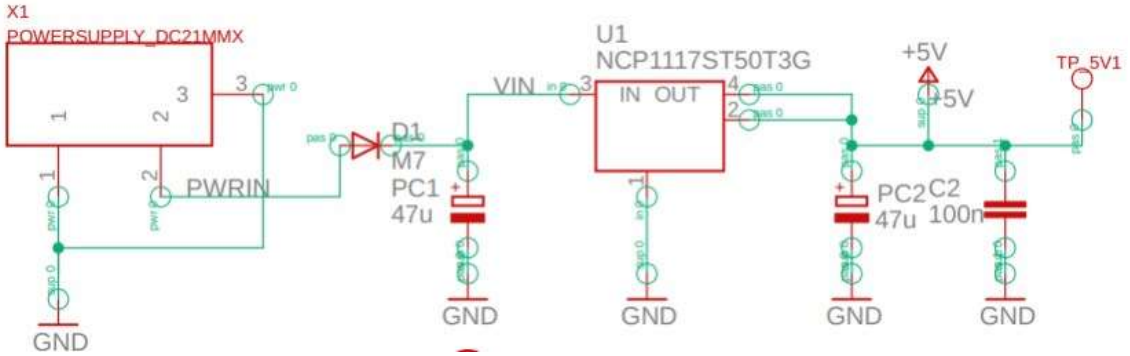


Nombre	Valor
Tipo de memoria de programa	Flash
Tamaño de memoria de programa	32Kb
Velocidad de la CPU	20 MIPS
SRAM	2048 B
EEPROM	1024 bytes
Comunicación Serie	1-UART, 2-SPI, 1-I2C
Captura/comparación/PWM	1-CCP/ 6-PWM
Timers	2 x 8 bits

LAYOUT ARDUINO UNO



Entrada de alimentación por conector DC: Esta entrada de alimentación se encuentra conectada directamente a la entrada del regulador de tensión interno de 5V de la placa Arduino UNO a través de un condensador estabilizador de tensión. Esta entrada puede ser alimentada con voltajes no regulados con valores comprendidos entre los 7V y los 12V



Puerto de alimentación: El puerto de alimentación consta de un conector tipo Ribbon de 6 pines que podemos emplear para alimentar tanto la propia placa Arduino UNO, como otros sensores y dispositivos, siempre y cuando el consumo total sea menor a 1A.

- Vin: Se encuentra conectado a la entrada de regulador de tensión interno de 5V. Podemos emplearlo para alimentar la tarjeta Arduino Uno de la misma forma que por medio del conector DC.
- GND: Masa de la tarjeta Arduino UNO.
- 5V: punto de salida del regulador de tensión interno de 5V. Podemos emplear este pin para alimentar otros dispositivos que funcionen a 5V o alimentar la propia tarjeta Arduino UNO a través de este pin si disponemos de una fuente externa regulada de 5V.
- 3.3V: salida del regulador interno de la tarjeta Arduino UNO de 3,3V. Podemos emplear este pin para alimentar otros dispositivos que operen con 3,3V.
- Reset: Se encuentra físicamente interconectado al pin número 1 del microcontrolador ATMEGA 328P a través de una resistencia de PULL-UP. Puede emplearse para reiniciar el microcontrolador si se impone un valor de 0 lógico (GND).

También existe la posibilidad de alimentar la tarjeta Arduino Uno a través del puerto de programación y comunicación USB. ***Hay que tener en cuenta que únicamente se puede alimentar la tarjeta Arduino UNO a través de una de las opciones disponibles.***

Entradas analógicas: Me permite leer un valor de analógico de tensión con una resolución de 10 bits. El valor de tensión máximo de la señal analógica que deseamos medir no podrá exceder en ningún caso el valor de 5V.

Entradas y salidas digitales: Me permiten leer un valor digital de un sensor o activar y desactivar salidas. Los pines identificados con el símbolo "~" representan pines que pueden generar señales tipo PWM (3, 5, 6, 9, 10, 11).

LED BUILTIN: diodo led que se encuentra físicamente conectado al pin digital 13 y que podemos encender o apagar configurando este pin como salida y activándola o desactivándola.

ESTRUCTURA DE UN PROGRAMA

El entorno de desarrollo de Arduino ha sido diseñado para evitar que el programador tenga que realizar las configuraciones necesarias para poner en funcionamiento el microcontrolador, en su lugar ha relegado la estructura elemental de un programa a dos funciones básicas o mínimas que deben aparecer en cualquier programa que diseñemos orientado a estas tarjetas de desarrollo. Las funciones elementales que debe contener cualquier programa desarrollado en Arduino son la función sin retorno setup() y la función sin retorno de valor loop(). Naturalmente como en cualquier otro programa podemos desarrollar nuestras funciones particulares.

```
void setup(){
    /*Código
    que deseamos
    ejecutar*/
}

void loop(){
    /*Código
    que deseamos
    ejecutar*/
}
```

Debemos tener en cuenta que el lenguaje implementado en Arduino es KeySensitive.

Por directiva no tenemos la obligación de rellenar con código ninguna de las dos funciones anteriores, sin embargo, deben aparecer imperativamente.

void setup(): el programa que incluyamos dentro de esta función se ejecutará una única vez cuando se inicie el microcontrolador. Esta función se emplea habitualmente para incluir código que solo deseemos que se ejecute al arranque de microcontrolador, como por ejemplo configuraciones relacionadas con puertos de entrada y salida, comunicaciones, etc.ç

void loop(): las instrucciones que recojamos en el interior de esta función se ejecutarán de forma indefinida y cíclicamente. Habitualmente en esta función se encapsulan las instrucciones relacionadas con el programa principal de microcontrolador.

```
void setup(){
    //CONFIGURAMOS EL PIN DIGITAL 13 COMO SALIDA
    pinMode(13, OUTPUT);
}
void loop(){
    //Activamos la salida 13
    digitalWrite(13, HIGH);
    //Esperamos 5 segundos
    delay(5000);
    //desactivamos la salida 13
    digitalWrite(13, LOW);
    //Esperamos 5 segundos
    delay(5000);
}
```

JUEGO DE INSTRUCCIONES

pinMode(número de pin, tipo): permite configura un pin digital como entrada o como salida.

- Número de pin: número de pin digital (0….13)
- Tipo: especifica la configuración que deseamos para ese pin: INPUT, INPUT_PULLUP, OUTPUT

digitalRead(número de pin): permite obtener el valor instantáneo existente en un pin digital configurado como entrada.

- Número de pin: número de pin digital (0….13)
- Resultado: HIGH, LOW.
- Resultado: 1, 0.

digitalWrite(número de pin, valor): permite establecer un "1" o un "0" lógico (0V o 5V) en un pin digital configurado como salida.

- Número de pin: número de pin digital (0….13)
- Valor: valor que deseamos establecer para ese pin (HIGH, LOW, 1, 0)

analogRead(número de pin analógico): permite obtener el valor de tensión instantáneo presente en el pin analógico especificado como entrada.

- Numero de pin analógico: pin analógico del que deseamos leer el valor (A0……A5)
- Resultado: un valor entero comprendido entre 0 y 1023 proporcional a la tensión presente en el pin.

anaologWrite(número de pin PWM, duty cicle): permite generar una señal PWM de periodo fijo con un ciclo de trabajo determinado en uno de los pines habilitados para tal fin.

- Número de pin PWM: pin habilitado para generar una señal PWM configurado previamente como salida (3, 5, 6, 9, 10, 11).
- Duty cicle: número entero de 8 bits (0…255) que especifica el ciclo de trabajo deseado para señal PWM (0%....100%).

delay(cantidad de milisegundos): permite congelar el microcontrolador la cantidad de milisegundos especificada como parámetro de entrada.

millis(): me devuelve el número de milisegundos que han transcurrido desde que se inició el microcontrolador. Se almacena en una variable long por lo que su valor desbordará aproximadamente al cabo de 9h.

micros(): me devuelve el número de microsegundos que han transcurrido desde que se inició el microcontrolador. Se almacena en una variable long por lo que su valor desbordará aproximadamente al cabo de 1h.

min(x, y): calcula el valor mínimo entre dos valores x e y de cualquier tipo y devuelve este valor.

max(x, y): calcula el valor máximo entre dos valores x e y de cualquier tipo y devuelve este valor.

Map(valor, mínimo valor de entrada, máximo valor de entrada, mínimo valor de salida, máximo valor de salida): nos devuelve un valor de tipo long resultado de escalar el valor pasado como parámetro de entrada entre los nuevos límites especificados.

- Valor: valor que deseamos escalar.
- Mínimo valor de entrada: menor cantidad que puede tener el valor inicialmente.
- Máximo valor de entrada: mayor cantidad que puede tener el valor inicialmente.
- Mínimo valor de salida: menor cantidad que deseamos tenga el valor.
- Máximo valor de salida: mayor cantidad que deseamos tenga el valor.

Serial.begin(número de baudios): Permite inicializar la comunicación serie a través del puerto USB por medio de la unidad USART del microcontrolador. Como parámetro de entrada se le debe especificar la velocidad en baudios a la que deseamos implementar la comunicación serie.

- 300, 600, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200.

Serial.print(Cadena o valor): permite representar por el puerto serie USB la cadena de caracteres o el valor que le pasemos como parámetro de entrada sin incluir un retorno de carro.

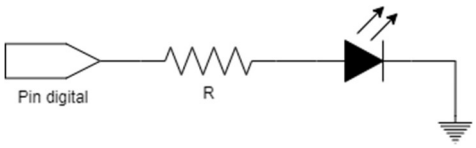
Serial.println(Cadena o valor): permite representar por el puerto serie USB la cadena de caracteres o el valor que le pasemos como parámetro de entrada incluyendo un retorno de carro.

Serial.available(): nos devuelve un valor TRUE siempre y cuando haya un dato disponible para leer desde el puerto serie USB.

Serial.read(): nos devuelve el último byte recibido por el puerto serie, habitualmente se suele recoger un valor tipo char que se corresponda con un carácter ASCII.

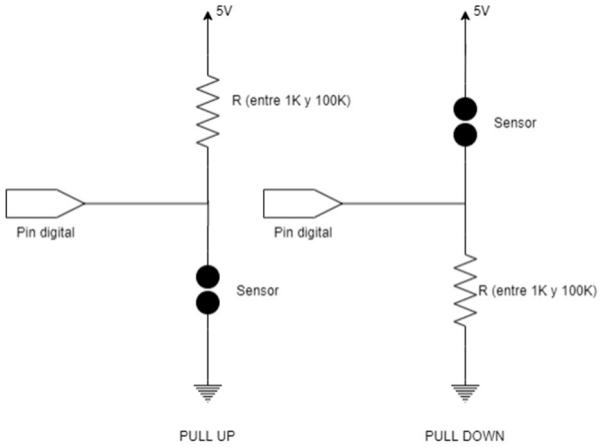
CONEXIONES ÚTILES

Activar un led



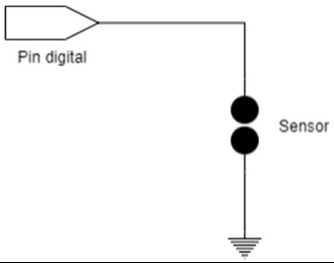

```
#define LED_PIN 6

void setup(){
    //CONFIGURAMOS EL PIN DIGITAL COMO SALIDA
    pinMode(LED_PIN, OUTPUT);
}

void loop(){
    //Activamos la salida
    digitalWrite(LED_PIN, HIGH);
    //desactivamos la salida
    digitalWrite(LED_PIN, LOW);
}
```

Lectura de una entrada digital libre de potencial



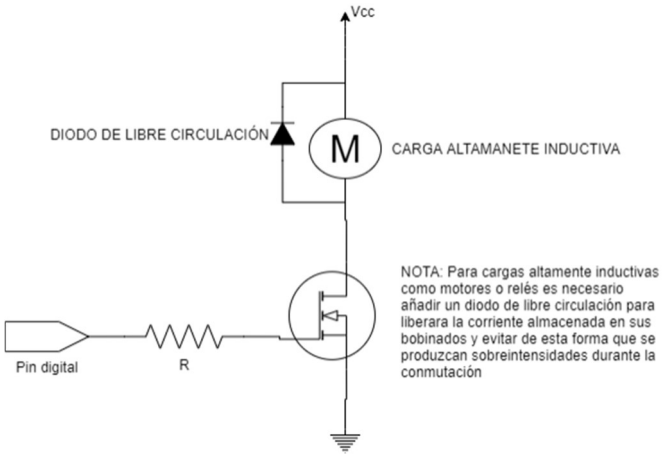

```
#define PIN 6

void setup(){
    pinMode(PIN, INPUT);
}

void loop(){
    int valor = digitalRead(PIN);
}
```

Lectura de una entrada digital (con pullup interno)



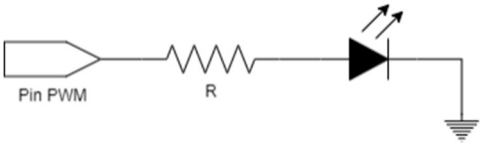

```
#define PIN 6

void setup(){
    pinMode(PIN, INPUT_PULLUP);
}

void loop(){
    int valor = digitalRead(PIN);
}
```

Controlar una salida digital con alto consumo



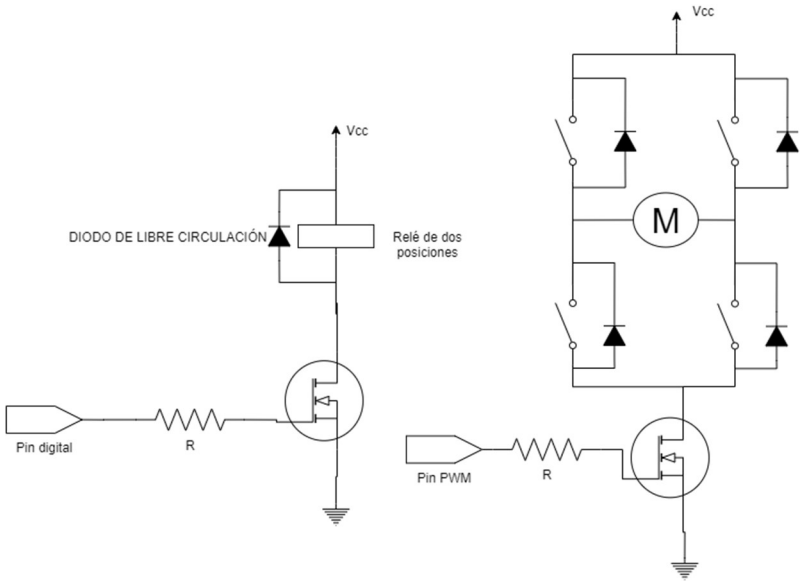

```
#define PIN 6

void setup(){
    //CONFIGURAMOS EL PIN DIGITAL COMO SALIDA
    pinMode(PIN, OUTPUT);
}

void loop(){
    //Activamos la salida
    digitalWrite(PIN, HIGH);
    //desactivamos la salida
    digitalWrite(PIN, LOW);
}
```

Modular el brillo de un led

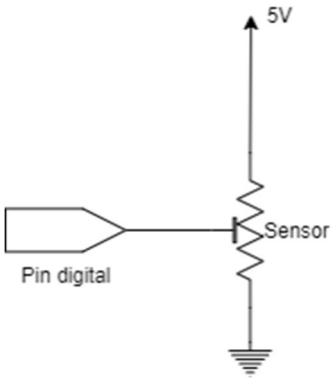



```
#define PIN_PWM 5

void setup(){
    pinMode(PIN_PWM, OUTPUT);
}

void loop(){
    for(int i= 0; i<256; i++){
        analogWrite(PIN_PWM, i);
        delay(5000);
    }
}
```

Cambio de giro de un motor y control de velocidad

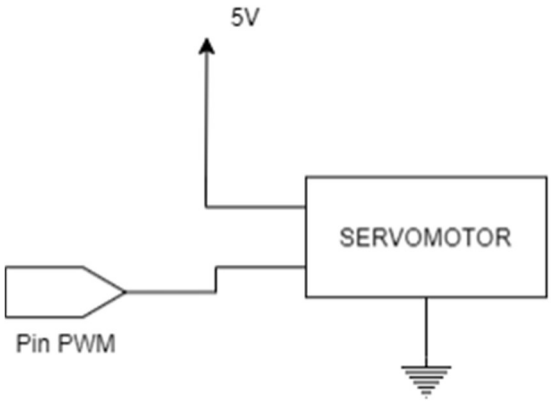


Lectura de una entrada analógica




```
#define ANALOG_PIN A0

void setup(){
    ///los pines analógicos nunca se congiguran
    /// son entradas SIEMPRE
}

void loop(){
    int value = analogRead(ANALOG_PIN);
    float tension = value * 5.0 / 1023.0;
}
```

Controlar un servomotor




```
#include <Servo.h>
#define PIN_PWM 5

Servo servomotor

void setup(){
    servomotor.attach(PIN_PWM);
}

void loop(){

    //le especificamos el ángulo que
    //queremos que gire en grados
    servomotor.write(90);
}
```

Temporización usando millis()

```
unsigned long timerStart, timerCurrent;
int interval = 1000;  /// 1000 ms

void setup(){
   timerStart = millis();
}

void loop(){
    timerCurrent = millis();
    if(timerCurrent - timerStart > interval){
        /* Código que
        deseo ejecutar
        */
       timerStart = millis();
    }
    timerCurrent = millis();
}
```